03

Actualizando datos #2

Transcripción

[00:00] Es básicamente porque aquí yo en ningún momento lo he dicho a Spring: “Spring, por si acaso, estos parámetros que tú ves aquí son los que están llegando del body al request”. ¿Recuerdan? Exacto, me está faltando RequestBody. Entonces yo voy a copiar nuevamente RequestBody aquí y le voy a decir que me lo valide, porque yo quiero que el id no llegue nulo.

[00:27] Entonces @RequestBody @Valid DatosActualizarMedico. ¿Por qué? Porque ahora Spring con eso ya sabe, voy a agarrar los parámetros que están aquí, porque si no simplemente este parámetro es nulo y id simplemente es nulo. Por eso es que me da ese tipo de excepción de ir a Access Exception.

[00:45] Vamos a esperar que el servidor cargue, me parece que ya recargó. Vamos a enviar otra vez y retornó 200. Vamos a ver si es verdad esto. Me voy a mi listado de médicos, vemos que mi id 5 está con Diego López, lo voy a ejecutar nuevamente y vemos que aún está con Diego López. Aquí sucedió algo. No funcionó el update.

[01:09] Y ustedes me preguntarán: “Diego, ¿por qué no funcionó el update?” Simple. En ningún momento estamos cerrando la transacción para que el médico actualice los datos, por transacción. ¿Se les ocurre o conocen una anotación para esos casos? Exacto, estamos hablando de @Transactional.

[01:31] Con @Transactional, yo voy a guardar aquí y automáticamente JPA Hibernate va a mapear que cuando termine este método la transacción se va a liberar. Ese es un concepto básico entre las acciones a nivel de base de datos,

inicia la transacción a este nivel y cuando termine lo que va a hacer es un commit de los datos a nivel ya de base de datos. Por lo tanto, mi transacción va a ejecutarse realmente.

[02:02] Voy a entonces darle un ActualizarMedico otra vez, voy a enviar, me da 200 y cuando listo mis médicos vemos que ahora ya Luis López ya fue actualizado con el id 5. ¿Por qué? Nuevamente Transactional lo que hace es abrir una transacción en la base de datos, JPA lo que hace es mapear mi médico que yo estoy trayendo de la base de datos, ahora ya lo tengo en el contexto de JPA.

[02:29] Y una vez que yo actualizo los datos de este médico, como está dentro de esta transacción, una vez que el método termina, la transacción también termina. Una vez que una transacción termina, hace un commit en la base de datos, y mis cambios son guardados ejecutados correctamente.

[02:48] Otro beneficio de la transacción es que, por ejemplo, si tú tienes múltiples actualizaciones por caso, yo quiero actualizar aquí los datos del médico y acá tengo otro método de actualizarDirección. ¿Qué pasaría si actualizo correctamente los datos del médico y sucede un error al momento de actualizar dirección?

[03:07] Los datos médicos ya estarían guardados y habría inconsistencia de datos. Con @Transactional eso ya está cubierto porque simplemente la transacción no va a hacer un commit en la base de datos. Va a hacer un rollback y no sucedió nada. Eso ya es algo un poco más profundo a nivel de lo que es Spring Boot, Spring JPA en sí y no va a ser parte del enfoque de este curso, pero es bueno que lo sepan.

[03:36] Viene mucho en su vida laboral diaria. @Transactional es una anotación que van a usarla regularmente, por ejemplo aquí, en el PostMapping también podríamos usar @Transactional, pero estamos llamando directamente al repository.save. En este caso aquí no estamos llamando a ningún repositorio

para hacer el update de los datos, solo estamos usando JPA puro. Por eso el @Transactional aquí es necesario.

[04:01] Solo queda una sola cosa más por hacer aquí y es el delete, pero eso es un tema del siguiente video.